PyGerber

Release 1.1.0

Krzysztof Wiśniewski

Oct 20, 2021

CONTENTS

PyGerber is a Python library for 2D and 3D rendering of Gerber X3 files. It is completely written in Python, and only dependencies are limiting its portability.

CHAPTER

ONE

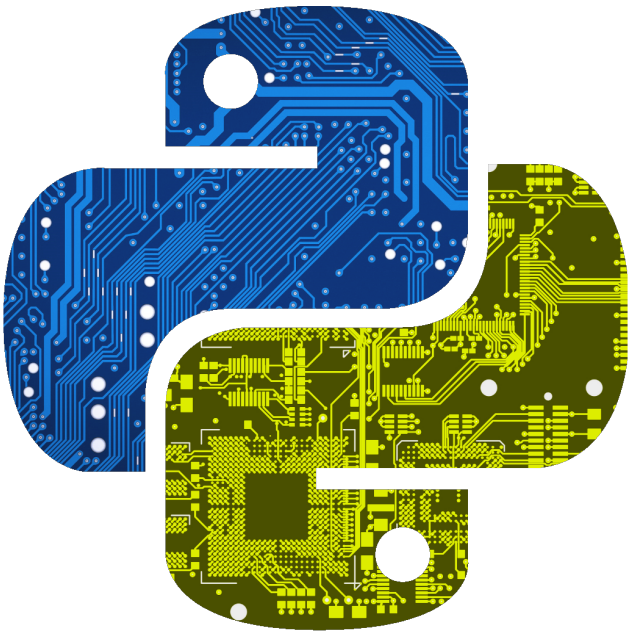
CONSTRUCTIVE CRITICISM

I would like to point out that this is the first extensive documentation I have ever written, so I would be very grateful for any constructive criticism.

CHAPTER

TWO

CONTENTS



2.1 Overview

PyGerber is a Python library for 2D and 3D rendering of Gerber X3 files. It is completely written in Python, and only dependencies are limiting its portability.

This package is a Free Software; it is released under MIT license. Be aware that dependencies might be using different licenses.

PyGerber offers a CLI and API for Python to allow easy rendering of Gerber files. Parser was build with GBR X3 format in mind, however, it has extensive support for older standards and deprecated features. Package is using third party libraries for low level drawing and mesh creation.

PyGerber's parser was not mend to be used by package users, but there are no obstacles preventing you from using it. However, stability of the API is not guaranteed between minor releases (I'll do my best to make it stable among patches).

2.1.1 Installation

PyGerber is available on PyPI and can be obtained via pip

```
pip install pygerber
```

You can also install the in-development version from github with

```
pip install https://github.com/Argmaster/pygerber/archive/main.zip
```

Blender dependency issue mentioned in previous releases was resolved by using PyR3 package which provides Blender. **However, blender has to be installed independently from package by calling PyR3.install_bpy script**:

```
python -m PyR3.install_bpy
```

Before You try to use 3D rendering.

2.1.2 Compatibility

PyGerber officially runs on Python 3.9.* and only on this version. However it may be possible to run 2D rendering on other Python versions that are supported by Pillow.

I'll consider bringing Python 3.8 3D rendering support, but no sooner than after implementation of full set of 3D rendering features and macros support.

2.1.3 Documentation

Documentation of this library is available at https://pygerber.readthedocs.io/

2.2 PyGerber Installation

PyGerber is available on PyPI and can be obtained via pip

```
pip install pygerber
```

You can also install the in-development version from github with

```
pip install https://github.com/Argmaster/pygerber/archive/main.zip
```

Blender dependency issue mentioned in previous releases was resolved by using PyR3 package which provides Blender. **However, blender has to be installed independently from package by calling PyR3.install_bpy script**:

```
python -m PyR3.install_bpy
```

Before You try to use 3D rendering.

2.3 Usage

PyGerber project provides both CLI and API for rendering Gerber files. For rendering projects consisting of multiple files, it is necessary to use so called specfiles, which describes project structure. Specfiles has different options for 2D and different for 3D rendering.

2.3.1 Command Line Interface

First and foremost, the PyGerber command line help page can be displayed with this command:

```
$ python -m pygerber -h
```

2D rendering Example

To render project in 2D, specified by "tests/gerber/pillow/specfile.yaml" (from our repo) and save it as PNG named "render.png" we can use following command:

```
$ python -m pygerber --pillow --toml "tests/gerber/pillow/specfile.yaml" -s "render.png"
```

YAML specfile used defines simple 4-layer PCB project and looks like this:

Listing 1: tests/gerber/pillow/specfile.yaml

```
dpi: 600
ignore_deprecated: yes
image_padding: 0
layers:
  - file_path: ./tests/gerber/set/top_copper.grb
```

(continues on next page)

(continued from previous page)

```
    colors:
      dark: [40, 143, 40, 255]
      clear: [60, 181, 60, 255]
  - file_path: ./tests/gerber/set/top_solder_mask.grb
    colors: solder_mask
  - file_path: ./tests/gerber/set/top_paste_mask.grb
  - file_path: ./tests/gerber/set/top_silk.grb
    colors: silk
```

On the very top level it specifies **DPI** of output image, **600** in our case, and sets **image padding** to **no padding**. Then **layers** param specifies layers from bottom-most to top-most. You have to at least specify path to gerber file in layer. For more about specfiles see *Specfile for 2D rendering* chapter.

3D rendering Example

3D rendering works identically, except that specfiles have a slightly different set of parameters. Again, see *Specfile for 3D rendering* chapter for more in-depth description. (DRY :D)

We also have example specfiles for 3D rendering in our repo, lets see how to use them

```
$ python -m pygerber --blender --toml "tests/gerber/blender/specfile.json" -s "render.glb
↪"
```

That's exactly the same 4 layer project, only this time in 3D and with specfile written in JSON instead of YAML

Listing 2: tests/gerber/blender/specfile.json

```
{
    "ignore_deprecated": true,
    "layers": [
        {
            "file_path": "./tests/gerber/set/top_copper.grb",
            "structure": {
                "material": {
                    "color": [40, 143, 40, 255]
                },
                "thickness": 0.78
            }
        },
        {
            "file_path": "./tests/gerber/set/top_solder_mask.grb",
            "material": "solder_mask"
        },
        {
            "file_path": "./tests/gerber/set/top_paste_mask.grb"
        },
        {
            "file_path": "./tests/gerber/set/top_silk.grb"
        }
    ]
}
```

As You can see, both top level options and layer specifications differ a bit, but remain similar. For the third time, I insist that you visit *Specfile for 3D rendering*. If you did, you should already be aware of the meaning of this file's structure.

2.3.2 2D rendering API

Coming soon. (v1.0.2)

2.3.3 3D rendering API

Coming soon. (v1.0.2)

2.3.4 Project Specification files

So called **specfiles contains description of project to be rendered**. The parameters that can be used in them differ depending on whether you are rendering in 2D or 3D Specfiles can be written in one of three languages: **JSON**, **YAML** and **TOML**. You have to indicated which one of them was used by using either corresponding flag, *(–yaml for YAML e.t.c)* when using CLI or appropriate function *(render_from_yaml() for YAML e.t.c)*.

Specfile for 2D rendering

2D spec top level parameters

At top level **specfile** contains a *dictionary* with following keys:

- `dpi` - integer, DPI of output image, *optional*, **defaults to 600**,
- `ignore_deprecated` - bool, if false, causes Gerber parser to halt after encountering deprecated syntax, *optional*, **defaults to True**,
- `layers` - list of layers, *mandatory*, each layer is a *dictionary* with following keys
  - `file_path`, string, path to Gerber source file, *mandatory*,
  - `colors`, *optional*, see *Defining colors for layer*

Defining colors for layer

`colors` layer dictionary param can be omitted, then color will be determined from Gerber file name. Chosen color will be one of *Predefined Colors*, the first whose name will be found in the filename.

`colors` can also be set to *string*, which have to be one of *Predefined Colors*.

Third option is to set colors manually via *dictionary*, then dictionary has following keys, whose values are lists of 3 or 4 integers in range 0-255 representing RGB / RGBA colors

- `dark` - *mandatory*, no default value
- `clear` - *optional*, defaults to transparent
- `background` *optional*, defaults to transparent

Predefined Colors

Following predefined colors are available:

- `silk` white, transparent, transparent
- `paste_mask` gray, transparent, transparent
- `solder_mask` light gray, transparent, transparent
- `copper` green, green (lighter), transparent
- `orange` orange, transparent, transparent
- `green` green, green (darker), transparent
- `debug` strange, strange, transparent

Example JSON specfile

Listing 3: tests/gerber/pillow/specfile.json

```
{
    "dpi": 600,
    "ignore_deprecated": true,
    "image_padding": 0,
    "layers": [
        {
            "file_path": "./tests/gerber/set/top_copper.grb",
            "colors": {
                "dark": [40, 143, 40, 255],
                "clear": [60, 181, 60, 255]
            }
        },
        {
            "file_path": "./tests/gerber/set/top_solder_mask.grb",
            "colors": "solder_mask"
        },
        {
            "file_path": "./tests/gerber/set/top_paste_mask.grb"
        },
        {
            "file_path": "./tests/gerber/set/top_silk.grb",
            "colors": [[255, 255, 255, 255]]
        }
    ]
}
```

Specfile for 3D rendering

3D spec top level parameters

At top level specfile contains a dictionary with following keys:

- `ignore_deprecated` - bool, if false, causes Gerber parser to halt after encountering deprecated syntax, *optional*, **defaults to True**,
- `scale` - float, output scale, *optional*, **defaults to 1000**, not yet modifiable.
- `layers` - list of layers, *mandatory*, each layer is a *dictionary* with following keys
  - `file_path`, string, path to Gerber source file, *mandatory*,
  - `structure`, *optional*, see *Defining structure of a layer*

Defining structure of a layer

`structure` parameter can be omitted, then structure will be determined from Gerber file name. Chosen structure will be one of *Predefined layer structures*, the first whose name will be found in the filename.

Otherwise it can be a string containing name of one of *Predefined layer structures*.

Last possible option is to use a dictionary with following keys:

- `material` - dictionary with blender BSDF node parameters. Only exception from typical node parameters is that colors are lists integers in range 0-255, **not** floats 0.0 - 1.0
- `thickness` - thickness of layer in millimeters as float.

Predefined layer structures

Following predefined layer structure are available:

- `silk` 0.04mm, rough, non-metalic, white
- `paste_mask` 0.1mm, metallic, partially rough, gray
- `solder_mask` 0.1mm, metallic, partially rough, gray (lighter)
- `copper` 0.78mm, metallic, rough, green
- `green` 0.78mm, default, green
- `debug` 0.78mm, default, strange
- `debug2` 0.78mm, default, strange
- `debug3` 0.78mm, default, strange

Example YAML specfile

Listing 4: tests/gerber/blender/specfile.yaml

```yaml
ignore_deprecated: yes
layers:
  - file_path: ./tests/gerber/set/top_copper.grb
    structure:
      thickness: 0.78
      material:
        color: [40, 143, 40, 255]
  - file_path: ./tests/gerber/set/top_solder_mask.grb
    material: solder_mask
  - file_path: ./tests/gerber/set/top_paste_mask.grb
  - file_path: ./tests/gerber/set/top_silk.grb
```

2.4 pygerber

2.4.1 pygerber package

Subpackages

pygerber.parser package

Subpackages

pygerber.parser.blender package

Submodules

pygerber.parser.blender.api module

pygerber.parser.blender.cli module

pygerber.parser.blender.parser module

Module contents

pygerber.parser.pillow package

Subpackages

pygerber.parser.pillow.apertures package

Submodules

pygerber.parser.pillow.apertures.arc_mixin module

class pygerber.parser.pillow.apertures.arc_mixin.**ArcUtilMixinPillow**
Bases: *pygerber.renderer.arc_util_mixin.ArcUtilMixin*

dpmm: float

get_arc_traverse_step_angle(*begin_angle*, *end_angle*, *radius*)

pygerber.parser.pillow.apertures.circle module

class pygerber.parser.pillow.apertures.circle.**PillowCircle**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)
Bases: *pygerber.parser.pillow.apertures.arc_mixin.ArcUtilMixinPillow*, *pygerber.parser.pillow.apertures.flash_mixin.FlashUtilMixin*, *pygerber.renderer.aperture.circular.CircularAperture*

arc(*spec:* pygerber.renderer.spec.ArcSpec) → None

property diameter: float

draw_shape(*aperture_stamp_draw: PIL.ImageDraw.Draw*, *color: Tuple*)

line(*spec:* pygerber.renderer.spec.LineSpec) → None

property radius: float

pygerber.parser.pillow.apertures.custom module

class pygerber.parser.pillow.apertures.custom.**PillowCustom**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)
Bases: *pygerber.parser.pillow.apertures.arc_mixin.ArcUtilMixinPillow*, *pygerber.parser.pillow.apertures.flash_mixin.FlashUtilMixin*, *pygerber.parser.pillow.apertures.flash_line_mixin.FlashLineMixin*, *pygerber.renderer.aperture.custom.CustomAperture*

pygerber.parser.pillow.apertures.flash_line_mixin module

class pygerber.parser.pillow.apertures.flash_line_mixin.**FlashLineMixin**
Bases: `object`

line(*spec:* pygerber.renderer.spec.LineSpec) → None

pygerber.parser.pillow.apertures.flash_mixin module

class pygerber.parser.pillow.apertures.flash_mixin.**FlashUtilMixin**
Bases: *pygerber.parser.pillow.apertures.util.PillowUtilMethdos*

property aperture_mask: PIL.Image.Image

property aperture_stamp_clear: PIL.Image.Image

property aperture_stamp_dark: PIL.Image.Image

draw_shape(*aperture_stamp_draw: PIL.ImageDraw.Draw*, *color: Tuple*)

flash(*spec:* pygerber.renderer.spec.FlashSpec) → None

flash_at_location(*location:* pygerber.mathclasses.Vector2D) → None

flash_offset()

get_aperture_bbox() → Tuple[float]

get_aperture_hole_bbox() → *pygerber.mathclasses.BoundingBox*

property hole_diameter: float

property hole_radius: float

property pixel_bbox

pygerber.parser.pillow.apertures.obround module

class pygerber.parser.pillow.apertures.obround.**PillowObround**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.parser.pillow.apertures.flash_line_mixin.FlashLineMixin*, *pygerber.parser.pillow.apertures.rectangle.PillowRectangle*

draw_shape(*aperture_stamp_draw: PIL.ImageDraw.Draw*, *color: Tuple*)

pygerber.parser.pillow.apertures.polygon module

class pygerber.parser.pillow.apertures.polygon.**PillowPolygon**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.parser.pillow.apertures.arc_mixin.ArcUtilMixinPillow*, *pygerber.parser.pillow.apertures.flash_mixin.FlashUtilMixin*, *pygerber.parser.pillow.apertures.flash_line_mixin.FlashLineMixin*, *pygerber.renderer.aperture.polygon.PolygonAperture*

arc(*spec:* pygerber.renderer.spec.ArcSpec) → None

draw_shape(*aperture_stamp_draw: PIL.ImageDraw.ImageDraw*, *color: Tuple*)

property radius: float

pygerber.parser.pillow.apertures.rectangle module

class pygerber.parser.pillow.apertures.rectangle.**PillowRectangle**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.parser.pillow.apertures.arc_mixin.ArcUtilMixinPillow*, *pygerber.parser.pillow.apertures.flash_mixin.FlashUtilMixin*, *pygerber.renderer.aperture.rectangular.RectangularAperture*

arc(*spec:* pygerber.renderer.spec.ArcSpec) → None

draw_shape(*aperture_stamp_draw: PIL.ImageDraw.Draw*, *color: Tuple*)

line(*spec:* pygerber.renderer.spec.LineSpec) → None

property x_half: float

property y_half: float

pygerber.parser.pillow.apertures.region module

class pygerber.parser.pillow.apertures.region.**PillowRegion**(*renderer*)

Bases: *pygerber.parser.pillow.apertures.arc_mixin.ArcUtilMixinPillow*, *pygerber.renderer.aperture.region.RegionApertureManager*, *pygerber.parser.pillow.apertures.util.PillowUtilMethdos*

finish(*bounds: List[*pygerber.renderer.spec.LineSpec*]*) → None

pygerber.parser.pillow.apertures.util module

class pygerber.parser.pillow.apertures.util.**PillowUtilMethdos**

Bases: object

property canvas

property dpmm

property draw_canvas

get_clear_color()

get_color()

get_dark_color()

is_clear()

prepare_arc_spec(*spec:* pygerber.renderer.spec.ArcSpec) → *pygerber.renderer.spec.ArcSpec*

prepare_coordinates(*vector:* pygerber.mathclasses.Vector2D) → *pygerber.mathclasses.Vector2D*

prepare_flash_spec(*spec:* pygerber.renderer.spec.FlashSpec) → *pygerber.renderer.spec.FlashSpec*

prepare_line_spec(*spec:* pygerber.renderer.spec.LineSpec) → *pygerber.renderer.spec.LineSpec*

renderer: *pygerber.renderer.Renderer*

Module contents

Submodules

pygerber.parser.pillow.api module

class pygerber.parser.pillow.api.**PillowLayerSpec**(*file_path: 'str'*, *colors: 'ColorSet'*)

Bases: *pygerber.parser.project_spec.LayerSpecBase*

colors: *pygerber.parser.pillow.parser.ColorSet*

file_path: str

classmethod load(*contents: Dict*)

class pygerber.parser.pillow.api.**PillowProjectSpec**(*init_spec: Dict*)

Bases: *pygerber.parser.project_spec.ProjectSpecBase*

property LayerSpecClass: *pygerber.parser.pillow.api.PillowLayerSpec*

dpi: int = 600

ignore_deprecated: bool = True

image_padding: int = 0

layers: List[*pygerber.parser.pillow.api.PillowLayerSpec*] = []

render() → PIL.Image.Image

pygerber.parser.pillow.cli module

pygerber.parser.pillow.cli.**handle_pillow_cli**(*args*)

pygerber.parser.pillow.parser module

class pygerber.parser.pillow.parser.**ColorSet**(*dark: 'Color_Type'*, *clear: 'Color_Type' = (0, 0, 0, 0)*, *background: 'Color_Type' = (0, 0, 0, 0)*)

Bases: object

background: Tuple[float, float, float, float] = (0, 0, 0, 0)

clear: Tuple[float, float, float, float] = (0, 0, 0, 0)

dark: Tuple[float, float, float, float]

exception pygerber.parser.pillow.parser.**ImageSizeNullError**

Bases: IndexError

class pygerber.parser.pillow.parser.**ParserWithPillow**(**, ignore_deprecated: bool = True*, *dpi: int = 600*, *colors: pygerber.parser.pillow.parser.ColorSet = ColorSet(dark=(66, 166, 66, 255), clear=(16, 66, 36, 255), background=(0, 0, 0, 0))*, *image_padding: int = 0*)

Bases: *pygerber.parser.parser.AbstractParser*

apertureSet: ApertureSet = ApertureSet(circle=<class 'pygerber.parser.pillow.apertures.circle.PillowCircle'>, rectangle=<class 'pygerber.parser.pillow.apertures.rectangle.PillowRectangle'>, obround=<class 'pygerber.parser.pillow.apertures.obround.PillowObround'>, polygon=<class 'pygerber.parser.pillow.apertures.polygon.PillowPolygon'>, custom=<class 'pygerber.parser.pillow.apertures.custom.PillowCustom'>, region=<class 'pygerber.parser.pillow.apertures.region.PillowRegion'>)

property canvas: PIL.Image.Image

get_image() → PIL.Image.Image

is_rendered: bool

save(*file_path: str*, *format: Optional[str] = None*) → None

Saves rendered image. *file_path* A filename (string), pathlib.Path object or file object. *format* Optional format override. If omitted, the format to use is determined from the filename extension. If a file object was used instead of a filename, this parameter should always be used.

Module contents

Submodules

pygerber.parser.parser module

class pygerber.parser.parser.**AbstractParser**(*ignore_deprecated: bool = True*)
Bases: abc.ABC

apertureSet: *pygerber.renderer.apertureset.ApertureSet*

parse(*source_string: str*, *file_path: str = '<string>'*)

parse_file(*file_path: str*)

abstract save(*file_path: str*, *format: Optional[str] = None*) → None

tokenizer: *pygerber.tokenizer.Tokenizer*

pygerber.parser.project_spec module

class pygerber.parser.project_spec.**LayerSpecBase**
Bases: abc.ABC

abstract static load(*contents: Dict*) → *pygerber.parser.project_spec.LayerSpecBase*

class pygerber.parser.project_spec.**ProjectSpecBase**(*init_spec: Dict*)
Bases: abc.ABC

abstract property LayerSpecClass: *pygerber.parser.project_spec.LayerSpecBase*

classmethod from_json(*file_path: str*) → *pygerber.parser.project_spec.ProjectSpecBase*

classmethod from_toml(*file_path: str*) → *pygerber.parser.project_spec.ProjectSpecBase*

classmethod from_yaml(*file_path: str*) → *pygerber.parser.project_spec.ProjectSpecBase*

abstract render() → Optional[Any]

Module contents

pygerber.renderer package

Subpackages

pygerber.renderer.aperture package

Submodules

pygerber.renderer.aperture.aperture module

class pygerber.renderer.aperture.aperture.**Aperture**(*args: ADD_Token.ARGS*, *renderer: Renderer*)
Bases: abc.ABC, *pygerber.renderer.arc_util_mixin.ArcUtilMixin*

abstract arc(*spec:* pygerber.renderer.spec.ArcSpec) → None

arc_bbox(*spec:* pygerber.renderer.spec.ArcSpec) → *pygerber.mathclasses.BoundingBox*

abstract bbox() → *pygerber.mathclasses.BoundingBox*

abstract flash(*spec:* pygerber.renderer.spec.FlashSpec) → None

flash_bbox(*spec:* pygerber.renderer.spec.FlashSpec) → *pygerber.mathclasses.BoundingBox*

abstract line(*spec:* pygerber.renderer.spec.LineSpec) → None

line_bbox(*spec:* pygerber.renderer.spec.LineSpec) → *pygerber.mathclasses.BoundingBox*

pygerber.renderer.aperture.circular module

class pygerber.renderer.aperture.circular.**CircularAperture**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.renderer.aperture.aperture.Aperture*

DIAMETER: float

HOLE_DIAMETER: float

bbox() → *pygerber.mathclasses.BoundingBox*

pygerber.renderer.aperture.custom module

class pygerber.renderer.aperture.custom.**CustomAperture**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.renderer.aperture.aperture.Aperture*

process_args()

pygerber.renderer.aperture.polygon module

class pygerber.renderer.aperture.polygon.**PolygonAperture**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.renderer.aperture.circular.CircularAperture*

DIAMETER: float

HOLE_DIAMETER: float

ROTATION: float

VERTICES: float

pygerber.renderer.aperture.rectangular module

class pygerber.renderer.aperture.rectangular.**RectangularAperture**(*args: <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object at 0x7fc1b3825490>*, *renderer*)

Bases: *pygerber.renderer.aperture.aperture.Aperture*

HOLE_DIAMETER: float

X: float

Y: float

bbox() → *pygerber.mathclasses.BoundingBox*

pygerber.renderer.aperture.region module

class pygerber.renderer.aperture.region.**RegionApertureManager**(*renderer*)

Bases: abc.ABC, *pygerber.renderer.arc_util_mixin.ArcUtilMixin*

bbox(*bounds: List[*pygerber.renderer.spec.Spec*]*) → *pygerber.mathclasses.BoundingBox*

abstract finish(*bounds: List[*pygerber.renderer.spec.Spec*]*) → None

steps: List[Tuple[*pygerber.renderer.aperture.aperture.Aperture*, *pygerber.renderer.spec.Spec*]]

Module contents

Submodules

pygerber.renderer.aperture_manager module

class pygerber.renderer.aperture_manager.**ApertureManager**(*apertureSet: ApertureSet*, *renderer: Renderer*)

Bases: object

apertureSet: ApertureSet

apertures: Dict[int, Aperture]

current_aperture: Aperture = None

define_aperture(*type: str*, *name: str*, *ID: int*, *args: object*)

getApertureClass(*name: Optional[str] = None*, *is_region: bool = False*)

get_aperture(*id: int*) → Aperture

get_current_aperture()

renderer: Renderer

select_aperture(*id: int*)

set_defaults()

pygerber.renderer.apertureset module

class pygerber.renderer.apertureset.**ApertureSet**(*circle: Type[pygerber.renderer.aperture.aperture.Aperture], rectangle: Type[pygerber.renderer.aperture.aperture.Aperture], obround: Type[pygerber.renderer.aperture.aperture.Aperture], polygon: Type[pygerber.renderer.aperture.aperture.Aperture], custom: Type[pygerber.renderer.aperture.aperture.Aperture], region: Type[pygerber.renderer.aperture.region.RegionApertureManager]*)

Bases: object

circle: Type[*pygerber.renderer.aperture.aperture.Aperture*]

custom: Type[*pygerber.renderer.aperture.aperture.Aperture*]

getApertureClass(*name: Optional[str] = None*, *is_region: bool = False*) → *pygerber.renderer.aperture.aperture.Aperture*

obround: Type[*pygerber.renderer.aperture.aperture.Aperture*]

polygon: Type[*pygerber.renderer.aperture.aperture.Aperture*]

rectangle: Type[*pygerber.renderer.aperture.aperture.Aperture*]

region: Type[*pygerber.renderer.aperture.region.RegionApertureManager*]

pygerber.renderer.arc_util_mixin module

class pygerber.renderer.arc_util_mixin.**ArcUtilMixin**

Bases: object

static get_arc_co_functions(*radius*)

static get_arc_length(*radius*) → float

get_arc_points(*spec:* pygerber.renderer.spec.ArcSpec, *is_ccw: bool*) → *pygerber.mathclasses.Vector2D*

static get_arc_ratio(*relative_angle*)

get_arc_traverse_step_angle(*begin_angle*, *end_angle*, *radius*)

get_begin_end_angles(*spec:* pygerber.renderer.spec.ArcSpec)

static get_relative_angle(*begin_angle*, *end_angle*)

property isCCW

pygerber.renderer.spec module

class pygerber.renderer.spec.**ArcSpec**(*begin: 'Vector2D'*, *end: 'Vector2D'*, *center: 'Vector2D'*, *is_region: 'bool' = False*)

Bases: *pygerber.renderer.spec.Spec*

bbox(*aperture:* pygerber.renderer.aperture.aperture.Aperture)

begin: *pygerber.mathclasses.Vector2D*

center: *pygerber.mathclasses.Vector2D*

draw(*aperture:* pygerber.renderer.aperture.aperture.Aperture)

end: *pygerber.mathclasses.Vector2D*

get_radius()

is_region: bool = False

class pygerber.renderer.spec.**FlashSpec**(*location: 'Vector2D'*, *is_region: 'bool' = False*)

Bases: *pygerber.renderer.spec.Spec*

bbox(*aperture:* pygerber.renderer.aperture.aperture.Aperture)

draw(*aperture:* pygerber.renderer.aperture.aperture.Aperture)

is_region: bool = False

location: *pygerber.mathclasses.Vector2D*

class pygerber.renderer.spec.**LineSpec**(*begin: 'Vector2D'*, *end: 'Vector2D'*, *is_region: 'bool' = False*)

Bases: *pygerber.renderer.spec.Spec*

bbox(*aperture:* pygerber.renderer.aperture.aperture.Aperture)

begin: *pygerber.mathclasses.Vector2D*

draw(*aperture:* pygerber.renderer.aperture.aperture.Aperture)

end: *pygerber.mathclasses.Vector2D*

is_region: bool = False

class pygerber.renderer.spec.**Spec**

Bases: abc.ABC

abstract bbox(*aperture*)

abstract draw(*aperture*)

Module contents

class pygerber.renderer.**Renderer**(*apertureSet:* pygerber.renderer.apertureset.ApertureSet)

Bases: object

apertures: *pygerber.renderer.aperture_manager.ApertureManager*

bbox_arc(*end:* pygerber.mathclasses.Vector2D, *offset:* pygerber.mathclasses.Vector2D) → None

bbox_flash(*point:* pygerber.mathclasses.Vector2D) → *pygerber.mathclasses.BoundingBox*

bbox_interpolated(*end:* pygerber.mathclasses.Vector2D, *offset:* pygerber.mathclasses.Vector2D) → *pygerber.mathclasses.BoundingBox*

bbox_line(*end:* pygerber.mathclasses.Vector2D) → None

current_point: *pygerber.mathclasses.Vector2D*

define_aperture(**args*, ***kwargs*)

draw_arc(*end:* pygerber.mathclasses.Vector2D, *offset:* pygerber.mathclasses.Vector2D) → None

draw_flash(*point:* pygerber.mathclasses.Vector2D) → None

draw_interpolated(*end:* pygerber.mathclasses.Vector2D, *offset:* pygerber.mathclasses.Vector2D) → None

draw_line(*end:* pygerber.mathclasses.Vector2D) → None

end_region()

finish_drawing_region() → Tuple[*pygerber.renderer.aperture.region.RegionApertureManager*, List[*pygerber.renderer.spec.Spec*]]

isCCW()

move_pointer(*location:* pygerber.mathclasses.Vector2D) → None

region_bounds: List[*pygerber.renderer.spec.Spec*]

render(*token_stack: Deque[*pygerber.tokens.token.Token*]*) → None

replace_none_with_0(*vector:* pygerber.mathclasses.Vector2D)

replace_none_with_current(*vector:* pygerber.mathclasses.Vector2D)

select_aperture(*id: int*)

set_defaults()

state: *pygerber.drawing_state.DrawingState*

total_bounding_box(*token_stack: Deque[*pygerber.tokens.token.Token*]*)

pygerber.tokens package

Submodules

pygerber.tokens.add module

class pygerber.tokens.add.**ADD_Token**(*match_object: re.Match*, *state: DrawingState*)

Bases: *pygerber.tokens.token.Token*

ARGS = <pygerber.tokens.add.ADD_Token.ARGS_dispatcher object>

class ARGS_dispatcher(*pattern: str | Callable*)

Bases: *pygerber.validators.struct_validator.StructValidator*

DIAMETER = <pygerber.validators.coordinate.UnitFloat object>

HOLE_DIAMETER = <pygerber.validators.coordinate.UnitFloat object>

ROTATION = <pygerber.validators.basic.Float object>

VERTICES = <pygerber.validators.basic.Int object>

X = <pygerber.validators.coordinate.UnitFloat object>

Y = <pygerber.validators.coordinate.UnitFloat object>

re_match: re.Match

BASIC_APERTURE = '(?P<TYPE>[CROP]),(?P<ARGS>([-+]?[0-9]*\\.?[0-9]*X?)+)'

CIRCLE_PATTERN = re.compile('(?P<DIAMETER>[-+]?[0-9]*\\.?[0-9]*)(X(?P<HOLE_DIAMETER>[-+]?[0-9]*\\.?[0-9]*))?')

FLOAT_PATTERN = '[-+]?[0-9]*\\.?[0-9]*'

ID = <pygerber.validators.basic.Int object>

NAME = <pygerber.validators.basic.String object>

NAMED_APERTURE = '(?P<NAME>[a-zA-Z0-9]+)'

POLYGON_PATTERN = re.compile('(?P<DIAMETER>[-+]?[0-9]*\\.?[0-9]*)X(?P<VERTICES>[-+]?[0-9]*\\.?[0-9]*)(X(?P<ROTATION>[-+]?[0-9]*\\.?[0-9]*))?(X(?P<HOLE_DIAMETER>[-+]?[0-9]*\\.?[0-9]*))?')

RECTANGLE_PATTERN = re.compile('(?P<X>[-+]?[0-9]*\\.?[0-9]*)X(?P<Y>[-+]?[0-9]*\\.?[0-9]*)(X(?P<HOLE_DIAMETER>[-+]?[0-9]*\\.?[0-9]*))?')

TYPE = <pygerber.validators.basic.String object>

pre_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern = re.compile('%ADD(?P<ID>[0-9]+)((?P<TYPE>[CROP]),(?P<ARGS>([-+]?[0-9]*\\.?[0-9]*X?)+)|(?P<NAME>[a-zA-Z0-9]+))*%')

pygerber.tokens.am module

class pygerber.tokens.am.**ApertureMacro_Token**(*match_object: re.Match*, *state: DrawingState*)

Bases: *pygerber.tokens.token.Token*

BODY = <pygerber.validators.basic.String object>

NAME = <pygerber.validators.basic.String object>

re_match: re.Match

regex: re.Pattern = re.compile('%AM(?P<NAME>.*?)*(?P<BODY>.*?)*%', re.DOTALL)

pygerber.tokens.comment module

class pygerber.tokens.comment.**G04_Token**(*match_object: re.Match*, *state: DrawingState*)

Bases: *pygerber.tokens.token.Token*

STRING = <pygerber.validators.basic.String object>

re_match: re.Match

regex: re.Pattern = re.compile('G04(?P<STRING>.*?)*')

class pygerber.tokens.comment.**G74_Token**(*match_object: re.Match*, *state: DrawingState*)

Bases: *pygerber.tokens.token.Token*

re_match: re.Match

regex: re.Pattern = re.compile('G74*')

class pygerber.tokens.comment.**G75_Token**(*match_object: re.Match*, *state: DrawingState*)

Bases: *pygerber.tokens.token.Token*

re_match: re.Match

regex: re.Pattern = re.compile('G75*')

class pygerber.tokens.comment.**LoadName_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

re_match: re.Match

regex: re.Pattern = re.compile('%LN.*?*%', re.DOTALL)

pygerber.tokens.control module

class pygerber.tokens.control.**EndOfStream_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('M0[02]*')

class pygerber.tokens.control.**ImagePolarity_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

POLARITY = <pygerber.validators.basic.String object>

re_match: re.Match

regex: re.Pattern = re.compile('%IP(?P<POLARITY>((POS)|(NEG)))*%')

class pygerber.tokens.control.**Whitespace_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

keep: bool = False

regex: re.Pattern = re.compile('\\s+')

pygerber.tokens.dispatcher_meta module

class pygerber.tokens.dispatcher_meta.**Dispatcher**(*match_object: re.Match*, *state: DrawingState*)
Bases: object

Base class for all dispatcher objects. Includes tokens and some of the fields.

class pygerber.tokens.dispatcher_meta.**DispatcherMeta**(*name*, *bases*, *attributes*)
Bases: abc.ABCMeta

get_inherited_validators() → dict

get_validators() → dict

validators: Dict[str, *pygerber.validators.validator.Validator*] = {}

pygerber.tokens.dispatcher_meta.**getvalidators**(*mesh_factory:* pygerber.tokens.dispatcher_meta.Dispatcher) → dict
Returns validators specified for given Dispatcher.

Parameters **mesh_factory** (Dispatcher) – Object to fetch validators from.

Returns dictionary of factory fields.

Return type dict

pygerber.tokens.dnn module

class pygerber.tokens.dnn.**D01_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

I = <pygerber.validators.coordinate.Coordinate object>

J = <pygerber.validators.coordinate.Coordinate object>

X = <pygerber.validators.coordinate.Coordinate object>

Y = <pygerber.validators.coordinate.Coordinate object>

bbox(*renderer: Renderer*)

property end

property offset

post_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern = re.compile('(X(?P<X>[-+]?[0-9]+))?(Y(?P<Y>[-+]?[0-9]+))?(I(?P<I>[-+]?[0-9]+))?(J(?P<J>[-+]?[0-9]+))?D01*')

render(*renderer: Renderer*)
This method should be called only after token is dispatched and after alter_state().

class pygerber.tokens.dnn.**D02_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

X = <pygerber.validators.coordinate.Coordinate object>

Y = <pygerber.validators.coordinate.Coordinate object>

property point

post_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern = re.compile('(X(?P<X>[-+]?[0-9]+))?(Y(?P<Y>[-+]?[0-9]+))?D02*')

class pygerber.tokens.dnn.**D03_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.dnn.D02_Token*

bbox(*renderer: Renderer*)

post_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern = re.compile('(X(?P<X>[-+]?[0-9]+))?(Y(?P<Y>[-+]?[0-9]+))?D03*')

render(*renderer: Renderer*)
This method should be called only after token is dispatched and after alter_state().

class pygerber.tokens.dnn.**DNN_Loader_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

ID = <pygerber.validators.basic.Int object>

pre_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern = re.compile('D(?P<ID>[1-9][0-9]*)*')

class pygerber.tokens.dnn.G54DNN_Loader_Token(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.dnn.DNN_Loader_Token*

re_match: re.Match

regex: re.Pattern = re.compile('G54D(?P<ID>[1-9][0-9]*)*')

pygerber.tokens.fs module

class pygerber.tokens.fs.FormatSpecifierToken(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

property DEC_FORMAT

property INT_FORMAT

X_dec = <pygerber.validators.basic.Int object>

X_int = <pygerber.validators.basic.Int object>

Y_dec = <pygerber.validators.conditional.CallOnCondition object>

Y_int = <pygerber.validators.conditional.CallOnCondition object>

alter_state(*state: DrawingState*)
This method should be called before render().

property length

mode = <pygerber.validators.basic.String object>

re_match: re.Match

regex: re.Pattern = re.compile('%FS(?P<zeros>[LTD])(?P<mode>[AI])X(?
P<X_int>[1-6])(?P<X_dec>[1-6])Y(?P<Y_int>[1-6])(?P<Y_dec>[1-6])*%')

zeros = <pygerber.validators.basic.String object>

pygerber.tokens.gnn module

class pygerber.tokens.gnn.G0N_Token(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

INTERPOLATION = <pygerber.validators.basic.Int object>

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('G0(?P<INTERPOLATION>[1-3])*?')

class pygerber.tokens.gnn.G36_Token(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('G36*')

class pygerber.tokens.gnn.**G37_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

bbox(*renderer: Renderer*) → BoundingBox

post_render(*renderer: Renderer*)

pre_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern = re.compile('G37*')

render(*renderer: Renderer*)
This method should be called only after token is dispatched and after alter_state().

class pygerber.tokens.gnn.**G55_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

re_match: re.Match

regex: re.Pattern = re.compile('G55.*?*')

class pygerber.tokens.gnn.**G70_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('G70.*?*')

class pygerber.tokens.gnn.**G71_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('G71.*?*')

class pygerber.tokens.gnn.**G90_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('G90*')

class pygerber.tokens.gnn.**G91_Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('G91*')

pygerber.tokens.load module

class pygerber.tokens.load.**LoadMirroringToken**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

MIRRORING = <pygerber.validators.basic.String object>

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('%LM(?P<MIRRORING>((N)|(X)|(Y)|(XY)))*%')

class pygerber.tokens.load.**LoadPolarityToken**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

POLARITY = <pygerber.validators.basic.String object>

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('%LP(?P<POLARITY>[CD])*%')

class pygerber.tokens.load.**LoadRotationToken**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

FLOAT_PATTERN = '[-+]?[0-9]*\\.?[0-9]*'

ROTATION = <pygerber.validators.basic.Float object>

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('%LR(?P<ROTATION>[-+]?[0-9]*\\.?[0-9]*)*%')

class pygerber.tokens.load.**LoadScalingToken**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

FLOAT_PATTERN = '[-+]?[0-9]*\\.?[0-9]*'

SCALE = <pygerber.validators.basic.Float object>

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('%LS(?P<SCALE>[-+]?[0-9]*\\.?[0-9]*)*%')

class pygerber.tokens.load.**LoadUnitToken**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.token.Token*

UNIT = <pygerber.validators.basic.String object>

alter_state(*state: DrawingState*)
This method should be called before render().

re_match: re.Match

regex: re.Pattern = re.compile('%MO(?P<UNIT>(MM)|(IN))*%')

pygerber.tokens.token module

class pygerber.tokens.token.**Deprecated**(*message*)
Bases: object

class pygerber.tokens.token.**Token**(*match_object: re.Match*, *state: DrawingState*)
Bases: *pygerber.tokens.dispatcher_meta.Dispatcher*

alter_state(*state: DrawingState*)
This method should be called before render().

bbox(*renderer: Renderer*) → BoundingBox

keep: bool = True

post_render(*renderer: Renderer*)

pre_render(*renderer: Renderer*)

re_match: re.Match

regex: re.Pattern

render(*renderer: Renderer*)
This method should be called only after token is dispatched and after alter_state().

renderer: Renderer = None

Module contents

pygerber.validators package

Submodules

pygerber.validators.basic module

class pygerber.validators.basic.**Float**(*default: Optional[Any] = None*)
Bases: *pygerber.validators.validator.Validator*

class pygerber.validators.basic.**Function**(*function: Callable*)
Bases: *pygerber.validators.validator.Validator*

class pygerber.validators.basic.**Int**(*default: Optional[Any] = None*)
Bases: *pygerber.validators.validator.Validator*

class pygerber.validators.basic.**String**(*default: Optional[Any] = None*)
Bases: *pygerber.validators.validator.Validator*

pygerber.validators.conditional module

class pygerber.validators.conditional.**CallOnCondition**(*validator:* pygerber.validators.validator.Validator, *condition: callable*, *onfailure: callable*)

Bases: *pygerber.validators.validator.Validator*

pygerber.validators.coordinate module

class pygerber.validators.coordinate.**Coordinate**

Bases: *pygerber.validators.validator.Validator*

ensure_mm(*state:* pygerber.tokens.token.Token, *value: float*)

parse(*state: DrawingState*, *value: str*) → Any

class pygerber.validators.coordinate.**UnitFloat**(*default: Optional[float] = None*)

Bases: *pygerber.validators.coordinate.Coordinate*

pygerber.validators.struct_validator module

class pygerber.validators.struct_validator.**StructValidator**(*pattern: str | Callable*)

Bases: *pygerber.tokens.dispatcher_meta.Dispatcher*, *pygerber.validators.validator.Validator*

clean_args(*token: Token*, *state: DrawingState*, *value: str*)

empty_namespace(*token: Token*, *state: DrawingState*)

get_pattern(*token*, *value*) → re.Pattern

re_match: re.Match

pygerber.validators.validator module

class pygerber.validators.validator.**Validator**(*default: Optional[Any] = None*)

Bases: `object`

Module contents

Submodules

pygerber.API2D module

pygerber.API2D.**render_file**(*file_path: str*, ***, *dpi: int = 600*, *colors:* pygerber.parser.pillow.parser.ColorSet *= ColorSet(dark=(66, 166, 66, 255), clear=(16, 66, 36, 255), background=(0, 0, 0, 0))*, *ignore_deprecated: bool = True*, *image_padding: int = 0*) → PIL.Image.Image

Loads, parses and renders file from given path and returns its render as PIL.Image.Image.

Parameters

- **file_path** (*str*) – Path to gerber file to render.
- **dpi** (*int, optional*) – Output image DPI, defaults to 600

- **colors** (ColorSet, *optional*) – Color specification, defaults to DEFAULT_COLOR_SET_GREEN
- **ignore_deprecated** (*bool, optional*) – If false causes parser to stop when deprecated syntax is met, defaults to True
- **image_padding** (*int, optional*) – Additional image padding, defaults to 0

Returns Output image.

Return type Image.Image

pygerber.API2D.**render_file_and_save**(*file_path: str*, *save_path: str*, ***, *dpi: int = 600*, *colors: pygerber.parser.pillow.parser.ColorSet = ColorSet(dark=(66, 166, 66, 255), clear=(16, 66, 36, 255), background=(0, 0, 0, 0))*, *ignore_deprecated: bool = True*, *image_padding: int = 0*)

Loads, parses, renders file from file_path and saves it in save_path.

Parameters

- **file_path** (*str*) – Path to gerber file.
- **save_path** (*str*) – Path to save render.
- **dpi** (*int, optional*) – DPI of output image, defaults to 600
- **colors** (ColorSet, *optional*) – Color set to use, defaults to DEFAULT_COLOR_SET_GREEN
- **ignore_deprecated** (*bool, optional*) – If true, causes parser to not stop when deprecated syntax is found, defaults to True
- **image_padding** (*int, optional*) – Additional pixel padding for image, defaults to 0

pygerber.API2D.**render_from_json**(*file_path: str*) → PIL.Image.Image

Render 2D image from specfile written in json.

Parameters **file_path** (*str*) – json specfile path.

Returns rendered and merged image.

Return type Image.Image

pygerber.API2D.**render_from_spec**(*spec: Dict[str, Any]*) → PIL.Image.Image

Render 2D image from specfile alike dictionary.

Parameters **spec** (*Dict*) – specfile parameters dictionary.

Returns rendered and merged image.

Return type Image.Image

pygerber.API2D.**render_from_toml**(*file_path: str*) → PIL.Image.Image

Render 2D image from specfile written in toml.

Parameters **file_path** (*str*) – toml specfile path.

Returns rendered and merged image.

Return type Image.Image

pygerber.API2D.**render_from_yaml**(*file_path: str*) → PIL.Image.Image

Render 2D image from specfile written in yaml.

Parameters **file_path** (*str*) – yaml specfile path.

Returns rendered and merged image.

Return type Image.Image

pygerber.API3D module

pygerber.cli module

pygerber.cli.**get_argument_parser**() → argparse.ArgumentParser

pygerber.cli.**handle_blender_cli**(*_)

pygerber.cli.**handle_pygerber_cli**(*args*)

pygerber.constants module

class pygerber.constants.**Interpolation**
Bases: object

ClockwiseCircular = 2

CounterclockwiseCircular = 3

Linear = 1

class pygerber.constants.**Mirroring**
Bases: object

No = 'N'

X = 'X'

XY = 'XY'

Y = 'Y'

class pygerber.constants.**Polarity**
Bases: object

CLEAR = 'C'

DARK = 'D'

class pygerber.constants.**Unit**
Bases: object

INCHES = 'IN'

MILLIMETERS = 'MM'

pygerber.coparser module

class pygerber.coparser.**CoParser**
Bases: object

default_format = '%FSLAX36Y36*%'

dump(*co: float*) → str

format: *pygerber.tokens.fs.FormatSpecifierToken*

format_zeros(*float_string*)

get_mode() → str

get_zeros() → None

parse(*float_string: str*) → float

set_default_format()

set_format(*format:* pygerber.tokens.fs.FormatSpecifierToken) → None

set_mode(*mode: str*) → None

set_zeros(*zeros: str*) → None

pygerber.drawing_state module

class pygerber.drawing_state.**DrawingState**

Bases: object

begin_region()

coparser: *pygerber.coparser.CoParser*

end_region()

interpolation: *pygerber.constants.Interpolation*

is_regionmode: bool

mirroring: str

parse_co(*float_string: str*)

polarity: str

rotation: float

scale: float

set_co_format(*fs: FormatSpecifierToken*)

set_defaults()

set_interpolation(*interpolation*)

set_mirroring(*mode*)

set_polarity(*polarity*)

set_rotation(*angle: float*)

set_scaling(*scale: float*)

set_unit(*unit*)

unit: *pygerber.constants.Unit*

pygerber.exceptions module

exception pygerber.exceptions.**ApertureSelectionError**
Bases: Exception

exception pygerber.exceptions.**DeprecatedSyntax**
Bases: *pygerber.exceptions.InvalidSyntaxError*

exception pygerber.exceptions.**EndOfStream**
Bases: Exception

exception pygerber.exceptions.**FeatureNotSupportedError**
Bases: Exception

exception pygerber.exceptions.**InvalidCommandFormat**
Bases: *pygerber.exceptions.InvalidSyntaxError*

exception pygerber.exceptions.**InvalidSyntaxError**
Bases: Exception

exception pygerber.exceptions.**RenderingError**
Bases: Exception

exception pygerber.exceptions.**TokenNotFound**
Bases: *pygerber.exceptions.InvalidSyntaxError*

pygerber.exceptions.**suppress_context**(*exc: Exception*) → Exception

pygerber.mathclasses module

class pygerber.mathclasses.**BoundingBox**(*x0: 'float'*, *y0: 'float'*, *x1: 'float'*, *y1: 'float'*) → 'None'
Bases: object

as_tuple() → Tuple[float]
Tuple (left, upper, right, lower)

as_tuple_y_inverse() → Tuple[float]

contains(*other:* pygerber.mathclasses.BoundingBox) → bool

height() → float

include_point(*point:* pygerber.mathclasses.Vector2D) → *pygerber.mathclasses.BoundingBox*

left: float

lower: float

padded(*delta*) → *pygerber.mathclasses.BoundingBox*

right: float

transform(*vector:* pygerber.mathclasses.Vector2D) → *pygerber.mathclasses.BoundingBox*

upper: float

width() → float

class pygerber.mathclasses.**Vector2D**(*x: 'float'*, *y: 'float'*)
Bases: object

as_tuple()

as_tuple_3D(*z: float = 0.0*)

dot(*other:* pygerber.mathclasses.Vector2D) → float

floor()

length()

normalize()

x: **float**

y: **float**

pygerber.mathclasses.**angle_from_zero**(*vector:* pygerber.mathclasses.Vector2D) → float

pygerber.mathclasses.**format_bytes**(*val: float*) → str

pygerber.tokenizer module

class pygerber.tokenizer.**Tokenizer**(*ignore_deprecated: bool = True*)

Bases: object

begin_index: **int = 0**

char_index = 0

line_index = 1

push_token(*token: Token*) → None

raise_token_not_found(*source*)

set_defaults()

state: *pygerber.drawing_state.DrawingState*

token_stack: **collections.deque**

token_stack_size: **int = 0**

tokenize(*source*, *file_path: str = '<string>'*) → Deque[Token]

tokenize_file(*file_path: str | Path*) → Deque[Token]

Module contents

2.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

2.5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.5.2 Documentation improvements

PyGerber could always use more documentation, whether as part of the official PyGerber docs, in docstrings, or even on the web in blog posts, articles, and such.

2.5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Argmaster/pygerber/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

2.5.4 Development

To set up *pygerber* for local development:

1. Fork pygerber (look for the "Fork" button).
2. Clone your fork locally:

   ```
   git clone git@github.com:YOURGITHUBNAME/pygerber.git
   ```

3. Create a branch for local development:

   ```
   git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

   ```
   tox
   ```

5. Commit your changes and push your branch to GitHub:

   ```
   git add .
   git commit -m "Your detailed description of your changes."
   git push origin name-of-your-bugfix-or-feature
   ```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

2.6 Authors

- Krzysztof Wiśniewski - https://github.com/Argmaster

2.7 Changelog

2.7.1 0.0.0 (2021-09-25)

- First release on PyPI.
- Added 2D rendering

2.7.2 1.0.0 (2021-10-06)

- Added 3D rendering (not full-featured)
- Added CLI and API for 3D rendering
- Installing bpy is required for 3D rendering

2.7.3 1.0.1 (2021-10-08)

- Added promised documentation for CLI
- Added promised documentation for project specfiles
- Fixed RTD python version requirement building issue
- Updated devlopment pipeline

2.8 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

A

B

C

D

E

F

G

H

N

O

P

R

S

T

U

V

W

X

Y

Z